1. Introduction

This technical note describes a stream-based interface to packets and data frames. It is not restricted to database access.

Originally it was envisaged that separate mechanisms would be used to access packets and data frames. It became evident that there were enough similarities in the requirements to combine the two, thus avoiding duplication of effort.

Note that the current implementation is incomplete and only contains certain elements of this specification.

2. Purpose

This interface specifies a one-way (read-only) protocol for homogeneous access to packets and data frames. It is not appropriate for clients wishing to update these objects. Neither is it a generic object server. The concept is based on data frames being the starting point for data processing, and therefore providing a convenient method for their access. As noted above, the requirements for packets are similar.

The intention is that objects can be streamed from a variety of sources, including over networks. Clients are not required to have a Versant installation (though they will be able to do more if they do). This approach also allows the possibility to design IA as an internet-aware application.

3. Scope

This interface applies to all mission phases, not just ILT, in accordance with the concept of smooth transition.

It includes:

- Getting them from a local database.
- Getting them from a database across a network, including via the Internet.
- Getting them from local (non-Versant) persistent storage.
- Getting them from the telemetry ingestor.
- Getting them from the router.

It does not include:

- Any access to objects other than packets or data frames. However, *some* elements of the design *might* prove to be applicable to other objects.
- Any means of writing to any persistent storage, including the database.

However, it does not *preclude* normal database operations *if* the client is a Versant one. In other words, it imposes no restrictions on trusted database clients.

4. Requirements

This section is extracted from the output of Action CSDT#5/2-2, compiled by Kevin Galloway.

4.1 Purpose

To identify the telemetry and data frame services to be provided by the HCSS to the HCSS ILT clients.
Note: In the text below tsps = telemetry source packets = telemetry packets while pdfs = product data frames = data frames.

4.2 Scope

These requirements address only the telemetry packet and data frame needs of the ILT clients.
The requirements to write ILT-client logs etc to the database are *not* addressed here. The requirements presented here are the read-only requirements on tsps and pdfs.
The requirements of the telemetry ingestor, which writes tsps and pdfs to the HCSS archive are not presented here.
If other ILT-clients need tsp and pdf write capability then these requirements will need extending.

4.3 General Description

ILT clients will exist both within the HCSS (internal ILT clients) and outside the HCSS (external ILT clients). ILT clients need to be able to retrieve telemetry packets and data frames from the HCSS archive and to be able to receive telemetry packets and data frames in NRT. The HCSS must provide these services.

Examples of ILT clients are: RTA client, QLA client, IA, database browsers. Note that the telemetry ingestor can also be considered an ILT-client but its specific requirements are not addressed here.

Note: No assumption is made regarding whether RTA, QLA, IA, database browsers, ... are internal or external clients.

4.4 Capability requirements

Cap-Req-1: The HCSS shall provide tsps and pdfs to both internal and external ILT clients.

Cap-Req-2: ILT clients shall be able to request:

1. Near real time (NRT) access to tsps
2. NRT access to pdfs
3. Access to tsps in the database based on various selection criteria
4. Access to pdfs in the database based on various selection criteria

Comments associated with Cap-Req-2:
Note 1: PACS: Although PACS QLA works with dataframes it also needs housekeeping telemetry, TEI telemetry etc.

Note 2: PACS: RTA uses SCOS functionality

The selection criteria currently identified for the request types 3 and 4 above are:

i. Give me tsps/ pdfs as they arrive (near real-time connection)
ii. Give me tsps/ pdfs for specified time period
iii. Give me tsps/ pdfs associated with observation execution x
iv. Give me pdfs associated with building block y

Note 1: Bullet (iv) Currently all tsps are associated with an observation as not all packets contain, or have accessible, a BBID (see core class model).

The ILT-Client developers will be provided with an interface, which they can use to perform these requests.

Cap-Req-3: The interface used by the ILT-Clients shall be the same for all of the above request types.

Comments associated with Cap-Req-3:
The ILT-client developer simply selects the request type/ connection he wants. Irrespective of the request type the tsps and/ or the pdfs should be made available to the ILT-Client developer in the same way.

Cap-Req-4: It must be possible to serve data to a remote site, even through a firewall.

4.5 Constraint requirements

4.5.1 Access constraints

Con-Req-1: Access to the tsps and pdfs shall be read only.

Comments associated with Con-Req-1:
The writing of tsps and pdfs to the HCSS archive (by the telemetry ingestor) is outside the scope of these requirements

Con-req-2: Data rights (?) Are there any associated with ILT?

4.5.2 Operational constraints

TBW

4.5.3 Performance constraints

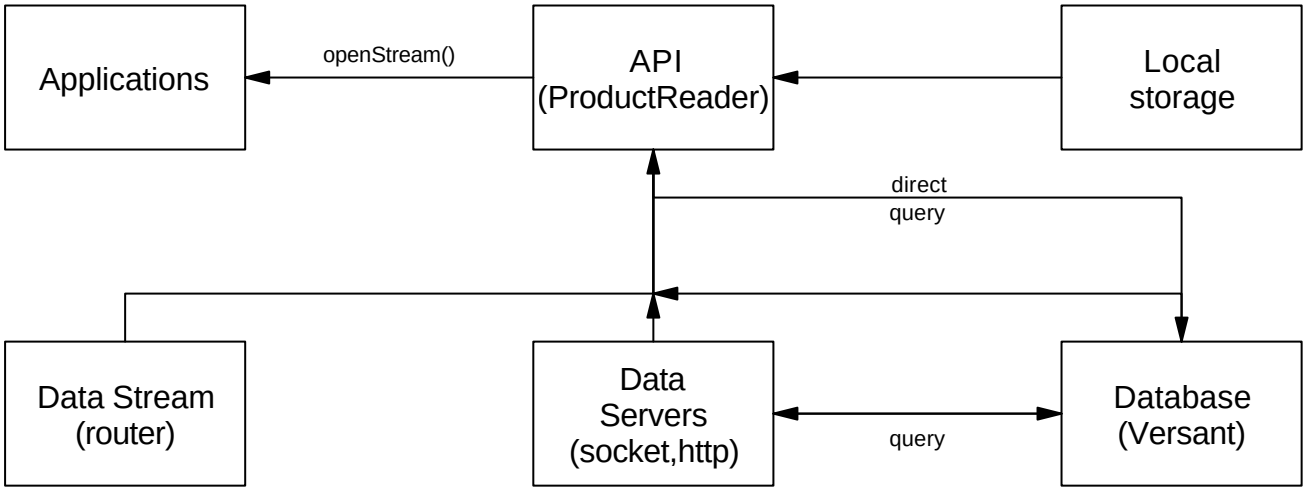
TBW

4.5.4 Reliability, maintainability and availability constraints

TBW

5. Architecture

The key to the architecture is the desire to abstract a connection to a stream of packet and data frame objects. Database access was originally conceived as a standard 3-tier architecture as is normal in the Java world. However the built-in flexibility makes supporting the need for 2-tier applications virtually trivial.



The Java implementation is split into connection and data storage packages in order to isolate the database-aware classes. The package organisation as it stands does not seem to be completely natural and should be reviewed.

5.1 Connection Layer

This layer is purely concerned with the mechanics of making a connection and is therefore mostly network-biased. It has no concept of persistent data storage. Note that the concept of a query is fully encapsulated by the HCSSAccess class. Some notes on recent changes are:

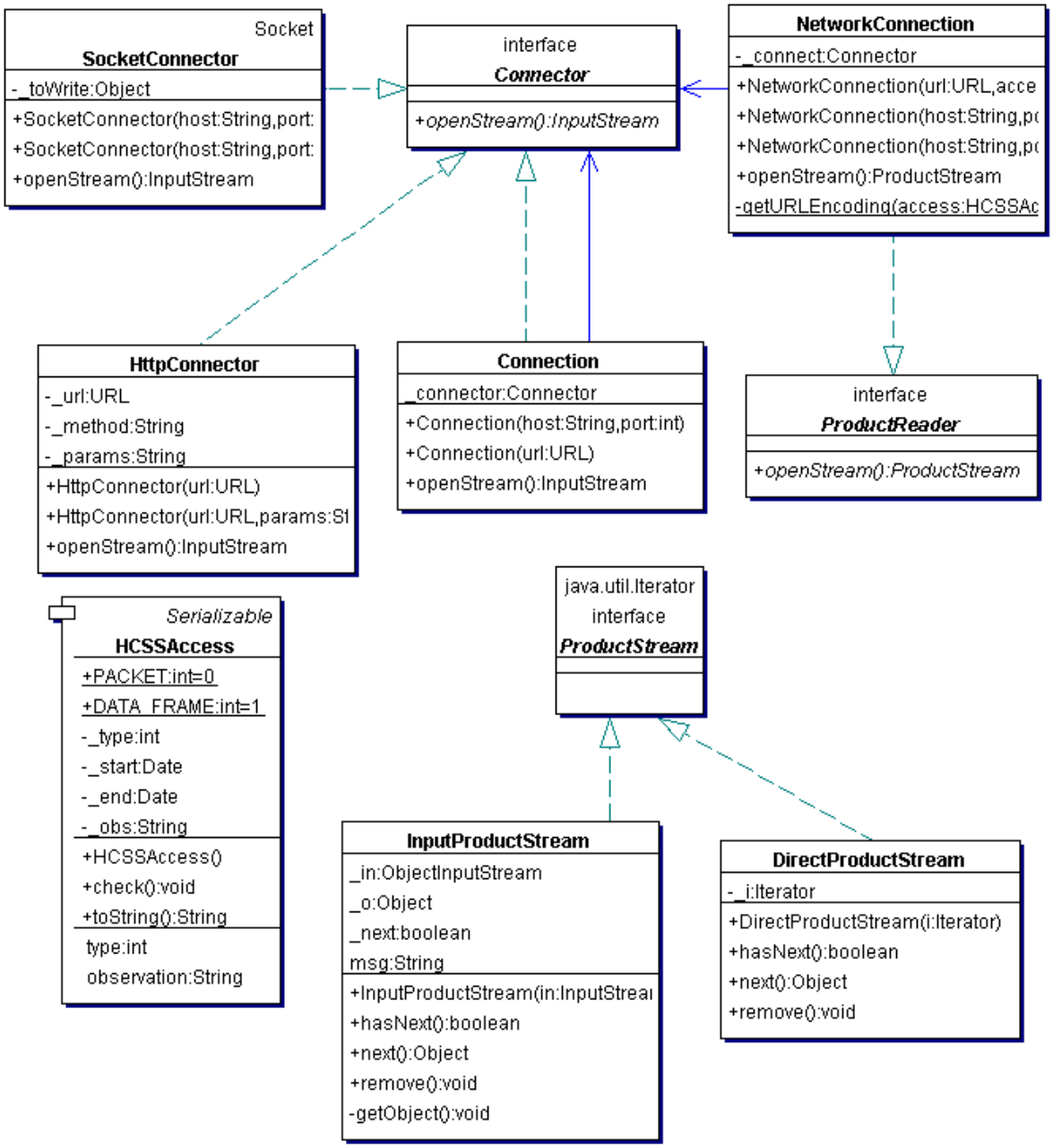
- HCSSConnection has been renamed to NetworkConnection to make its purpose and scope more explicit.
- ProductStream has been changed from a class to an interface.
- ProductReader has been added as a separate interface.

This package is currently missing a factory class to create ProductReader objects so that the type of implementation (local database, network, direct stream) is transparent to the application. The correct class could be dynamically loaded from a name in a properties file.

At some stage the issue of data rights will have to be addressed in the context of networked access.

An application uses the interface like this:

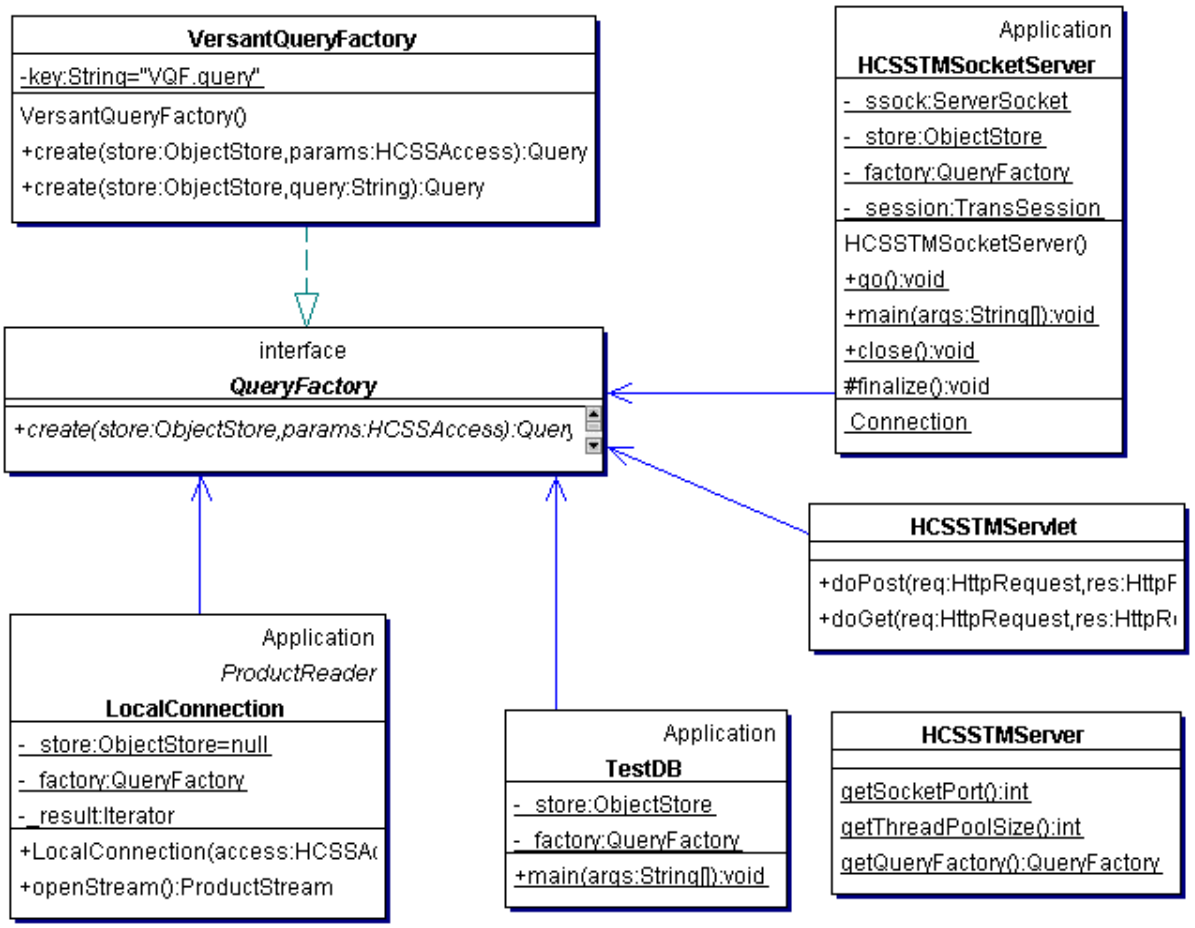
1. It constructs an HCSSAccess object to encapsulate the query. (This step is not necessary in applications where it makes no sense).
2. It creates an instance of ProductReader and opens a stream with the openStream() method.
3. It iterates through the resulting stream.



5.2 Data storage layer

This layer contains the classes for accessing the database. A LocalConnection class has been added here to support 2-tier implementations.

Note that QueryFactory objects are themselves created by a factory method by dynamically loading the correct class (set in a properties file). These query classes are intended to be the same as the ones in the 'store' package (though they don't currently have all the functionality).



6. Implications on database schema

6.1 Queries

I expect the natural and efficient way of finding these sorts of objects in the database will be by a database query (TBC). Database clients can potentially navigate from there. Queries will typically be on:

- Time
- Observation
- Building Block

This implies that these should all be fields that can be queried directly for both packets and data frames. In other words, both should tagged with the OBSID, BBID and a date field understandable by Versant (eg Date). It is possible to go directly to observation or building block and then navigate, but this is:

1. More complex.
2. Doesn't work with composite queries.
3. Due to serialization issues (see below), doesn't allow a non-Versant client to re-query (eg get frames for time range, then get the matching Observation objects).

It is not currently envisaged that PACS science telemetry will be tagged. This means that different instruments are likely to require different database search algorithms. Any such algorithms should be pluggable.

6.2 Serialization

Implications of serialization on the schema are applicable only to non-Versant clients. Any exporting of an object outside of Versant will cause it to be serialized. This will cause *all* references in the object to be recursively serialized with it. If the schema contains circular references this could result in the entire database being serialized. Possible solutions are:

- Don't allow objects to be exported. This is so restrictive as to be a non-starter.
- Have two core class models, one for internal (persistent) classes, and another external one for use in applications such as IA. This division could avoid a lot of confusion but would increase complexity. In practice it may not be necessary to duplicate the entire CCM. The external model could be considered as a 'view' of the database.
- Avoid unnecessary references in the schema. This compromises object navigation of the database. *This may be OK for packets and data frames though.*
- Implement readObject/writeObject methods so that references are not serialized. This has a drawback of leaving dangling references. Alternatively this could be hidden by façade classes (similar to the dual-CCM approach).
- Nullify references in the server. This is similar to the previous point.